HEWLETT PACKARD

# Automatic design of storage systems to meet availability requirements

*Khalil Amiri[1] and John Wilkes*

Storage Systems Program, Computer Systems Laboratory,
Hewlett-Packard Laboratories, Palo Alto, CA

**HPL–SSP–96–17**, 19 August 1996

*As storage systems continue to grow (due to the introduction of new IO interconnect such as FibreChannel), their design and configuration continues to be a costly and difficult task which involves complex trade-offs in cost, performance and availability. We focus on the problem of designing large storage systems to meet workload availability requirements (we use the term "workload unit" to refer to an object and the streams that access that object). We propose to reduce the difficulty of this problem by having end-users specify the availability requirements of workload units to the storage system and let the system configure itself to meet these requirements. We demonstrate the feasibility of this proposition by describing an approach and developing a working tool to automatically design storage systems to meet the availability requirements of a large set of workload units. The tool automatically synthesizes all candidate storage logical units that match the input workload units and assesses their reliability, availability and performance via automatically generated Markov chains. An Assignment engine selects the appropriate storage logical units and determines the assignment of workload units to storage logical units so as to minimize total storage system cost.*

[1]Khalil Amiri is a student at that Department of Electrical and Computer Engineering at Carnegie Mellon University, Pittsburgh, PA.

# 1 Introduction and motivation

Current storage systems require significant administration and high management costs. As storage systems continue to grow (due to fast IO interconnect such as FibreChannel), the problem of configuring and managing storage systems will even be more difficult. Storage systems composed of a connection of network-attached both simple and composite fault-tolerant storage devices present a wide choice of configurations that have varying availability and cost characteristics.

Our goal is to *automate* the design and configuration of storage systems to meet the *availability requirements* of applications. Applications specify their reliability and availability requirements to the storage system which automatically configures itself to meet them while minimizing total system cost. There are two main challenges to automating storage system design to meet availability requirements that we tackle in this paper:

1. How should the quality of service requirements of *workload units* be specified?

   We found out that the traditional availability measure of *mean availabilit*y does not provide the storage system with adequate information to determine the optimal configuration to match a set of workload units (we use a workload unit here to refer to an object together with the streams that access it). We develop a meaningful specification of QoS availability requirements that is *easy to use* (is based on information readily known to users) and that is *necessary* and *sufficient* to ensure that the storage system is optimally configured for the workload units' requirements.

2. How should the storage system be designed to meet the requirements of a large number of workload units while minimizing total system cost?

   Given sufficient knowledge about the application availability requirements, the storage system can in principle configure itself to meet these needs. We develop an approach and an algorithm implementation that performs automatic configuration.

The rest of the paper is organized as follows. In section 2, we review background information on reliability and availability. Section 3 describes the proposed specification of reliability and availability requirements. Section 4 describes the problem. Our solution approach is presented in section 5 and the conclusions are presented in section 6. Appendix A contains the mathematical analysis behind the availability computations.

# 2 Background: reliability and availability

Before diving into the details of our approach, we overview background information on reliability and availability. This section is only an overview and assumes that the reader is familiar with the main concepts in reliability and availability theories. However, this section can be skipped altogether and the reader should have no problem following the discussion in the rest of the paper.

## 2.1 Definitions

The *lifetime T* of a component is the time until the failure of the component, starting from some initial time (t=0) which corresponds to the time the component was installed. *T* is a random variable since it can not be predicted with certainty.

The reliability of a component is defined as the function R(t) which stands for the probability that the component is operational up and until time t. That is,

$$R(t) \equiv P(T > t)$$

In other words,

$$R(t) = P(T > t) = 1 - P(T < t) = 1 - F(t)$$

where F(t) is the cumulative distribution function of the component lifetime random variable *T*. The corresponding density function is denoted by *f(t)* which gives the probability that the lifetime is exactly *t*. Formally, *f(t) = P(T=t).*

The expected lifetime of a component E(T) can be computed to be

$$E(T) = \int_0^\infty t f(t)dt = \int_0^\infty [1 - F(t)]dt = \int_0^\infty R(t)dt$$

The failure rate of a component z(t) is the time dependent rate of failure of the component, and is defined by

$$z(t) \equiv \frac{f(t)}{1 - F(t)}$$

Informally, the failure rate at time *t* is the conditional probability that a component fails during the infinitely small interval $(t, t+\Delta t]$ given that it has survived until time *t* divided by the interval's length $\Delta t$.

## 2.2 Case of exponentially distributed lifetimes

Electronic components exhibit almost constant failure rates during their normal-life. Normal life is the period of time that follows an intial stage (after manufacturing) where a higher "infant mortality" failure rate is observed. Normal life is followed by a wear-out period where the failure rates increase again because of mechanical and electronic wear-out.

During normal life, if the failure rate z(t) is considered constant

$$z(t) = \lambda$$

Then the reliability function can be shown to have an exponential distribution [Siewiorek90], [Fleishmann96]

$$R(t) = e^{-\lambda t}$$

As a result, the expected lifetime E(T) can be computed to be

$$E(t) = \int_0^\infty R(t)dt = \int_0^\infty e^{-\lambda t}dt = \frac{1}{\lambda}$$

In this paper, we base some of our modeling and analysis on the assumption that storage system components exhibit constant failure rates (and therefore have exponentially distributed reliabilities).

The availability of a component is defined as the probability that the component is operating at time t. If the component is not repairable, then we obviously have *A(t) = R(t).* However, if the component fails and undergoes repair (during which it is restored to the operational state) then the two functions are different. The instantaneous availability as a function is hard to specify so a

continuous (interval) availability measure is usually used to characterize the availability of repairable systems. The continuous availability *A* of a component is defined as

$$A \equiv \lim_{t \to \infty} A(t)$$

## 2.3 Current reliability measures

We overview the traditional measures of system reliability and availability in this section.

The measure of reliability for a single component or a non-fault-tolerant system is *mean time to failure (MTTF)*. This is the expected lifetime of the component, which is equal to the inverse of the failure rate (in case of exponentially distributed reliability function). Similarly, the *mean time to repair (MTTR)* corresponds to the expected value of the repair time.

Continuous availability is equal to the proportion of time the component is not in repair taken over a long interval. It can be expressed in terms of the mean time to failure and mean time to repair as

$$A = \frac{MTTF}{MTTF + MTTR}$$

This relationship is based on the observation that the probability of being in an operational state is equal to the proportion of time the system is in the operational (non-repair) mode with respect to total time. *A* can therefore be expressed in terms of mean time between failures (MTBF) and mean time to repair as

$$A = \frac{MTBF - MTTR}{MTBF} = 1 - \frac{MTTR}{MTBF}$$

Figure 1. shows a diagram relating the mean reliability measures.

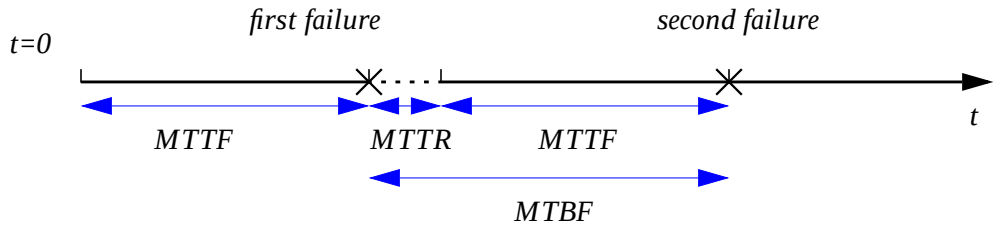


**Figure 1** Diagram of the mean reliability measures and their relationships.

## 2.4 New reliability measure: annualized failure rate

One problem with the mean time to failure (MTTF) or mean time between failures (MTBF) is that it is somewhat misleading to the general population of customers and system users. If the mean time to failure of a disk drive is 800,000 hours then the customer expects to operate without failures for that much time (91 years!).

In fact, electronic component and disk drives have a constant failure rate during their "normal-life." Most components are replaced before they start to exhibit "wear-out" related failures. Hence, the lifetimes (reliabilities) of such components follow an exponential distribution. If a component has an exponentially distributed reliability with parameter $\lambda$, then the reliability is expressed as

$$R(t) = e^{-\lambda t} = e^{-\left(\frac{1}{MTTF}\right)t}$$

Hence, the probability that a component survives till the expected lifetime (MTTF) is only 0.37

$$R(t)\big|_{t\,=\,MTTF} \;=\; e^{-1} \;=\; 0.37$$

Moreover, the median of the exponential distribution of parameter is 0.6931*MTTF. This means that 50% of the components fail before 69% of the MTTF specification period has elapsed.

Figure 2 illustrates the relationship between mean lifetime, median lifetime and reliability in case of the exponential distribution.

A better measure that Hewlett-Packard and the rest of the industry is shifting to is the *annualized failure rate*, or AFR[Allen96].

The annualized failure rate, or AFR, usually expressed as a percentage, predicts the number of failures that will occur over the course of a year for a certain population of components.

In the case of exponentially distributed reliability, the annualized failure rate is related to the mean time to failures as follows

$$AFR \;=\; 100 \times \frac{(HoursInOneYear)}{MTTF} \;=\; 100 \times \frac{356 \times 24}{MTTF} \;=\; 100 \times \frac{8760}{MTTF}$$

There are 356*24 hours a year, and assuming that the mean time to failure is much larger than a year, then the probability of a failure one year is the number of hours in a year multiplied by the failure rate (per hour). The failure rate is nothing but the inverse of the mean time to failure. This ratio is then multiplied by 100 to convert it to a percentage.

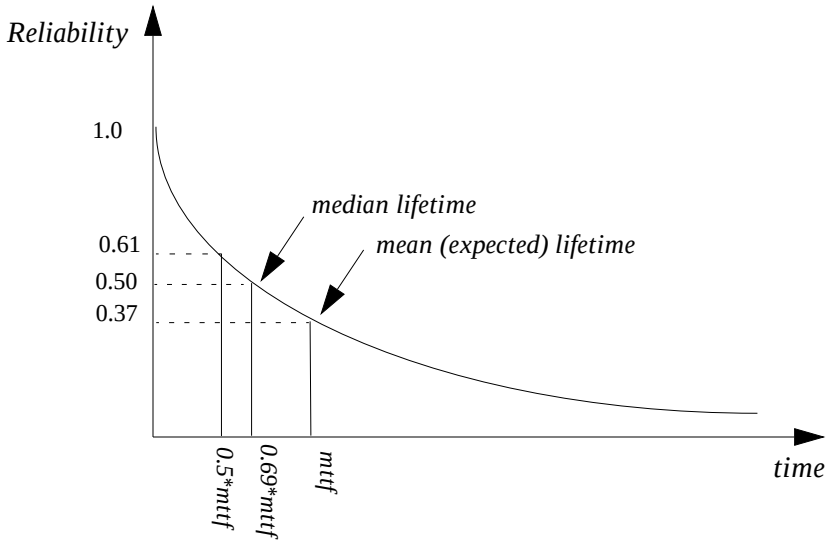


**Figure 2**. Exponential reliability versus time. There is a 37% chance of surviving until MTTF, a 61% chance of surviving half of the MTTF. 50% of the components are expected to fail before 0.69 of the MTTF has elapsed

## 2.5 Performability: performance aspects of reliable systems

The availability measure (as a proportion of time the system or component is available) is adequate for ultra-reliable systems, where there are two important states: failed and operational. Fault-tolerant computer systems (multiprocessor systems, storage systems) employ redundancy to improve reliability. However, this redundancy has important implications on performance.

During repair of a faulty component, a fault-tolerant system can be still available albeit at degraded performance levels. As shown in figure 2, performance can degrade below a certain application-dependent baseline performance. Real-time applications (video for instance) place a certain baseline performance requirement from the storage system (minimal data rate). The continuous availability measure *A* does not capture this degradation in performance and does not incorporate the notion of a minimal acceptable performance (baseline performance).

Previous work [Beaudry78], [Meyer82], [Pattipatti93] has addressed the performance aspects of reliable systems. The measure used to represent the cumulative performance of a fault-tolerant system as it undergoes degradation and repair has been coined "performability". The performability of a system with N configurations {1,2,3,...,N} is defined in [Pattipatti93], [Meyer82] as the probability density function of the stochastic process $y_t$

$$y_t = \int_0^t r_{x_\tau} d\tau$$

where

$$x_\tau \in \{1, 2, \ldots, N\}$$

is the discrete-valued state corresponding to the N possible system configurations and $r_{x_\tau}$ denotes the performance (reward or benefit) of the system when in state $x_\tau$ .

For our purposes, we define the average availability $a_i(t)$ of a storage system with respect to an object $w_i$ with baseline performance $bp_i$, over the interval $[0,t]$, to be the proportion of time the storage system provides the object with a higher than baseline performance.

$$a_i(t) = \frac{\int_0^t u[r_{x_\tau} - bp_i] d\tau}{t}$$

where u(t) is the step function defined over the set of real numbers as

$$u(t) = \begin{cases} 0, & t < 0 \\ 1, & t \geq 0 \end{cases}$$

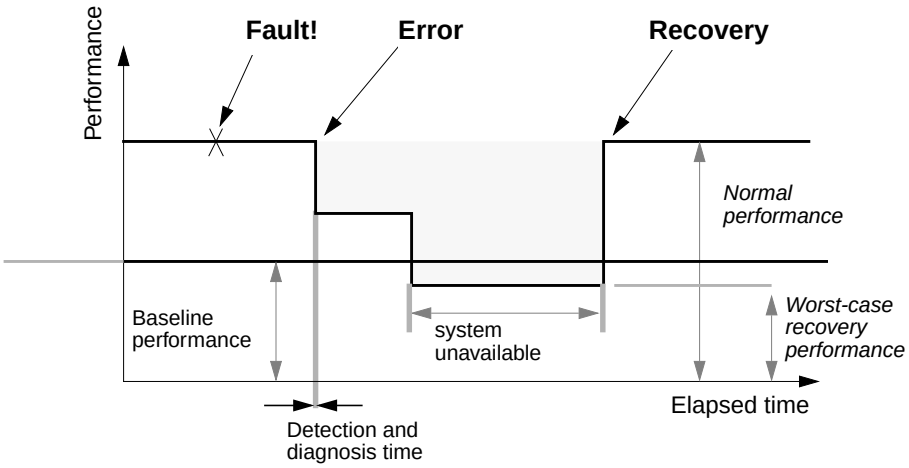


**Figure 3**. Faults and the performance of the system during recovery. The system is unavailable for the duration of time that its performance is below the baseline performance set by the application.

# 3 Specifying availability requirements

Self-configuring storage systems are based on the premise that applications specify their desired QoS requirements to the storage system, which configures itself to best meet them. In order for the storage system to determine the best possible configuration to match the applications, the QoS requirements should transfer as much information as possible about the application's data availability needs. In general, performance requirements are usually much better understood and therefore much more accurately specified than data availability requirements. This is partially due to the fact that the latter concepts are harder to grasp or measure in real life.

One fact remains, however, is that systems fail in predictable and measurable fashion, and have very predictable repair times. Consequently, the length of downtime and the frequency of occurrence of downtime can be accurately known for a given storage system.

There are two major trade-offs relating to data availability: a cost-availability trade-off (how much availability is worth the cost) and a performance-availability trade-off (how much performance is to be forsaken for higher availability). A good specification for availability goals should allow the application to express its needs as far as these two trade-offs are concerned.

This is a brief description of the availability requirements that we propose for workload units to specify:

- *Baseline performance requirements (BPR):* Minimal performance levels that are acceptable to the application (below which data is considered unavailable). Example: maximum response time of 100 msec.
- *Maximum write loss (MWL):* The maximum amount of writes (specified as a time period or size of updates) that can be lost without resulting in data loss as far as the application is concerned (this measure can be zero.) Example: the MWL=1day or 10KB (worth of updates.)
- *Annualized data loss rate (ADR):* The maximum tolerated probability of data loss over the course of a year. Data loss here occurs if data can not be recovered to a consistent state of updates within the MWL period specified by the application. Example: 0.001% annualized data loss rate.
- *Cost of data outage (CDO):* The cost of data outage specified as a function of the length of the outage. Example: a linear cost of data outage function of $10,000 per minute.
- *Mean data outage duration*

  The maximum data outage duration tolerated by the application.
- *Mean data outage rate*

  The mean annualized data outage rate (DOR) tolerated by the application.

  For a more detailed description of these QoS measures, refer to [Amiri96].

# 4 Problem description

Before describing the approach we take towards the solution, we describe the problem to give a feel of its complexity. Figure 4 represents a sketch of a networked storage system. We anticipate back-end interconnect technology such as FibreChannel and probably ATM to allow for a large number of heterogeneous network-attached devices to be interconnected together at client-access latencies that are comparable if not better than access latencies to local storage. FibreChannel and ATM lift the proximity and connectivity limitations of legacy storage interconnect technologies such as SCSI, which allows for wide-sharing and larger storage systems. Larger storage systems are substantially harder to design and configure. We believe storage systems can be automatically designed and configured to meet the reliability and availability requirements of applications.

Precisely, the problem is given a specification of the availability requirements of a set of workload units, how can we automatically configure the system to meet these requirements at minimal cost. In order to be able to identify the optimal (cost-effective) configuration and understand the space

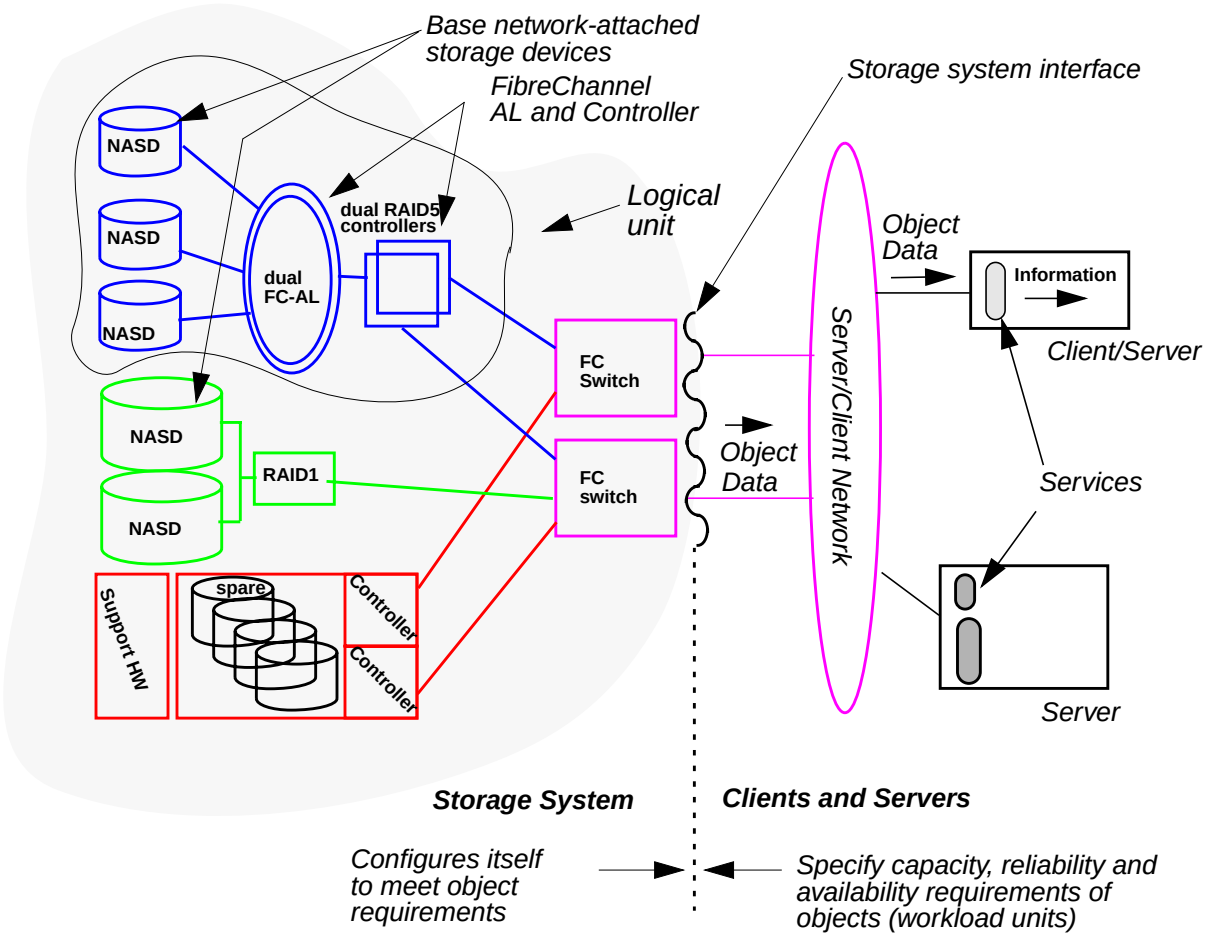


**Figure 4**. Storage systems based on network-attached peripherals consist of base storage devices, interconnect, controllers (caches) and support hardware (power supplies, fans). Demanding availability and reliability requirements are met via data replication and protection schemes (parity-based redundancy) as well as hardware redundancy schemes. A logical unit is a “virtual device” that is composed of physical devices and implements a logical data organization (such as a RAID5 data layout) and executes certain run-time policies (backup/recovery).

of possible configurations, we need to take a look at the architectural properties of storage subsystems that impact data availability.

## 4.1 Availability of storage systems

The availability of a storage subsystem is a function of several system properties. We break down the important properties that impact system availability to two categories: static and dynamic.

1. Static properties involve data layout schemes to tolerate storage device failures as well as hardware redundancy schemes to tolerate the failure of support hardware components. The data redundancy schemes we consider are parity-based RAID levels, strictly consistent and weakly consistent replication. Hardware redundancy involves the use of redundant power supplies, redundant controllers and dual interconnect links.
2. Dynamic properties involve the execution of special algorithms and policies. For example, backup and reconstruction (rebuild) algorithms and admission policies (what workload units are serviced in degraded modes -when some of the components have failed resulting in degraded performance).

## 4.2 Problem input

We assume that a pool of base storage devices and hardware components is available to build the system. These base components have different performance reliability and cost attributes.

We show below some sample input to the tool. The requirements of the workload units and the component specifications are described via attribute-value pairs. A Tcl script parses the input and creates the base devices and workload units and initializes their attributes. It then invokes the automatic design engine, which is implemented in C++, via an exciting set of macros (Tcl/C++ interface).

```
create_workload_unit w0
{{readThroughput 1e4} {writeThroughput 1e4}
 {minReadThroughput 5e3} {minWriteThroughput 5e3}
 {avgReqSize 4} {maxDataLossRate.0006}
 {maxDataOutageDuration 0.8} {maxDataOutageRate 12.2}
 {dataOutageCostModel 0} {dataOutageCostAlpha 1e2}
 {dataOutageCostBeta 10}
 {size 3000} {importance 1}};
create_simple_device disk0
{{modelRefNum 1}
 {SimpleDeviceReadThroughput 5e6}
{SimpleDeviceWriteThroughput 5e6}
 {SimpleDeviceMTTF 100e3}
 {SimpleDeviceTimeToReplace 0.5}
 {capacity 3000}
 {cost 450}};
```

# 5 Solution approach

The approach we take attempts to structure the search (design) space to simplify the task of automatic design. The approach uses the concept of a logical unit, which is a virtual or logical device that is composed of a set of base storage devices, a certain hardware architecture (controllers, power supplies, interconnect) and which has a set of dynamic properties (recovery algorithm, admission policy). Workload units are not assigned to physical devices, but rather to logical units. The cost, performance and availability characteristics of logical units can be computed from the characteristics of the base devices and from the data layout on the logical unit.

We first precisely define what constitutes a logical unit in the next section. Next, we overview the solution approach and discuss its details in the subsequent sections.

## 5.1 Logical units.

In this section, we define precisely what constitutes a logical unit. A logical unit is defined by:

1. a hardware organization:
   - **device_type**: The type of the simple base device (the type of the elementary disk drive).
   - **group_size**: a group size. The group size varies from a minimum minG=3 to a maximum size of maxG=10.
   - **num_spares**: number of spares {0 or 1}
   - **num_controllers**: number of controllers {1 or 2}
   - **num_supporthw_units**: number of support hardware units {1 or 2}
   - **num_loops**: number of local FibreChannel loops the devices are connected to {1 or 2}
2. a data layout
   - **replication_scheme**: can be one of the RAID redundancy schemes {RAID0, 1, 3, 4, 5}, or some form of strict or weakly-consistent replication.
   - **striping_choice**: the striping choice {non-striped, striped}
3. a set of policies

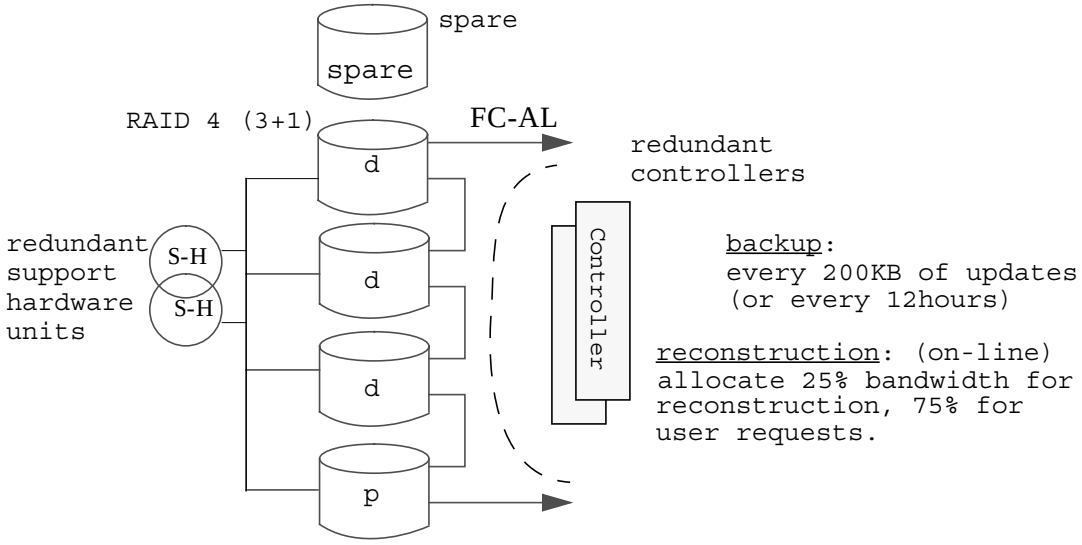


**Figure 5**. A logical unit is characterized by a hardware organization, a data layout and a set of policies.

- **`backup_policy:`** every *x* hours.
- **`cache_write_policy`**: write-through, write-back.
- **`recovery_policy`**: off-line (stop user requests), on-line (equal priority), on-line (recon priority).

## 5.2 Solution approach

Given a set of objects with specified availability requirements, the goal of this step is to integrate candidate logical units from a set of available storage devices and supporting components. This list of candidate logical units is then passed to the assignment engine which selects which logical units should be used and determines the minimal cost assignment of objects to logical units that meet the objects' requirements.

The approach we take can be divided into three steps:

1. Automatically synthesize all candidate logical units. A candidate logical unit is a logical unit that matches the requirements of at least one workload unit. Logical units that do not meet the requirements of any workload unit are discarded. Approximate analysis (analytic formulae) is used to estimate the characteristics of logical units at this step.
2. Model the candidate logical units to determine their reliability and availability characteristics (annualized data loss rate and the probability of being in each degraded mode together with the mean duration of the repair from each mode). Estimate the cost and performance of the logical units.
3. Determine a selection of logical units to be used and an assignment of workload units to the selected logical units that meets all requirements and minimized total cost.

## 5.3 Integration and availability modeling of logical units.

Logical units are automatically generated by integrating simple storage devices (disk drives), and other supporting components (controllers, support hardware). The available components and their characteristics (reliability and performance) are specified as input. For an arbitrary (automatically generated) logical unit, we are interested in estimating its annualized data loss rate and in computing the probability of the device being in each degraded mode. Degraded modes are modes where some of the components in the logical units have failed. A controller, disk or

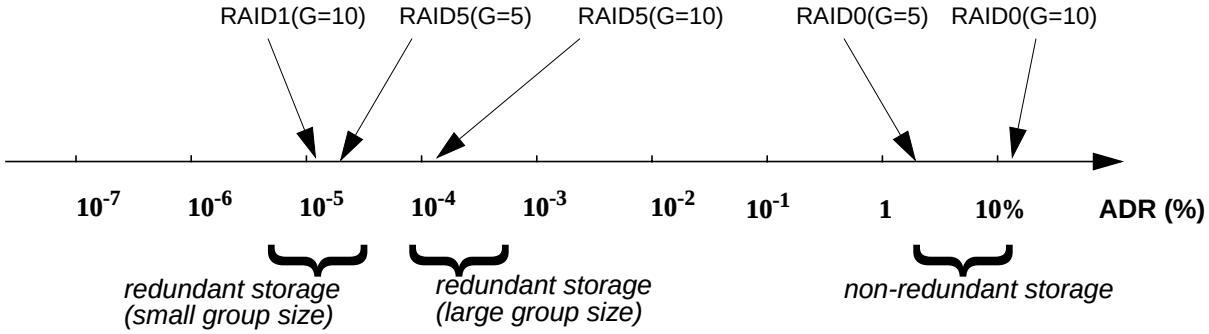


**Figure 6**. Annualized data loss rate (ADR) for alternative storage configurations. We assume that the groups are composed of homogenous disk drive (of MTTF=800,000 hours, AFR=1.095%). The data loss rates are computed approximately, assuming 1hour repair time for all fault-tolerant configurations (RAID1, RAID5) and assuming independent and exponentially distributed disk drive failures.

power failure causes the logical unit to go to a degraded mode. The logical unit may have either degraded performance (disk failure in a RAID5 array group) or zero performance (controller failure in a single controller disk array group). All such modes are referred to as degraded modes.

### *5.3.1 Estimating the annualized data loss rate*

There are several tools to estimate the reliability and availability of complex systems. These range from simple series-parallel models to fault trees, markov chains and stochastic petri nets [Siewiorek92], [Bavuso87]. Markov models represent a powerful tool for reliability and availability modeling because they can model several processes (such as failure and repair processes). A central concept to Markov models is the notion of a state and a state transition. The state of the system represents all relevant information needed to describe the state of each component in the system. For reliability (availability) modeling, a state is usually associated with each possible combination of failed and operational components in the system. Transitions from state to state represent the failure or repair of certain component. Continuous time markov chains are Markov models based on the premise that transitions can occur at any instant in time (continuous time model). A main assumption underlying Markov models is that the probability of a given state transition depends only on the current state, a nd not on the path the system travelled to get to that state. This memoryless (markovian) property, in the case of continuous time models, implies that the probability of a transition from a start state to some destination state is independent of the time spent already in the start state. This assumption implies that the waiting time until a transition is exponentially distributed. For reliability (availability) models of storage systems where the transitions correspond to component failures and repairs, the Markov model is well suited to the assumption that time to failure of disk drives and electronic component is exponentially distributed. Figure 6 is a partial sketch of a simple Markov chain used to model the reliability of a RAID5 disk array group with single power supply and dual controllers

A Markov chain is generated to model the reliability and availability of each logical unit. The markov chain is generated automatically from a description of the components in the unit, their failure and repair rates and from a definition of what constitutes a death state (data loss state) for the unit.

More precisely, a markov chain is generated from the following information about the logical unit:

1. The number of components in the logical unit:

   **(#disks, #controllers, #support_hardware_units,..., #local_interconnect_links)**

   specified as a tuple representing the number of components of each type in the unit.

2. The data loss states for this unit:

   **(max # of disk failures,..., max # of link failures)**

   Rather than specifying all the states that represent data loss state. This can be shorthanded as a tuple specifying the maximum number of failures of each type that can be tolerated before data loss (death state) is declared.

3. The failure and repair rates of each component, specified as a pair of tuples.

   **(disk failure rate, support hw failure rate,..., link failure rate)**

   **(disk repair rate, support hw repair rate,..., link repair rate)**

   As far as the implementation is concerned, the tuple is not a list of numbers but rather a list of functions. The function represents the failure rate of the component for a certain start state. Treating failure and repair rates as functions of the start state allow us to represent dependent failures among components.

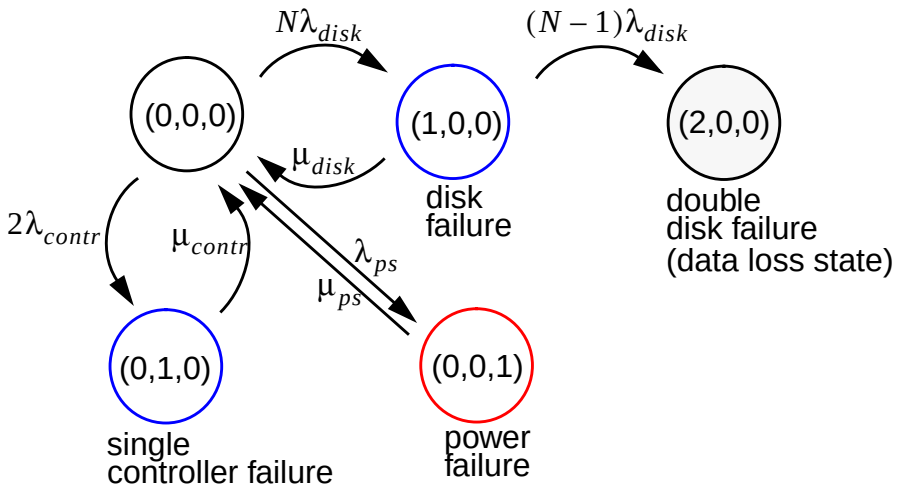


**Figure 7**. (Partial) Markov chain model for a logical unit consisting of dual disk controllers and N disks. State (0,0,0) is the initial operational state. State (1,0,0) occurs if one of the disks fails. State (2,0,0) is the data loss state (second disk failure before first is repaired). λ denotes the failure rate and μ the repair rate of components

### *5.3.2 Estimating degraded mode probabilities*

In addition to the mean time to data loss, we are interested in estimating the probability of the logical unit being in each degraded mode over the course of the use of the unit. The frequency of occurrence of degraded modes and their duration is important because it impacts the application perceived "data outage duration" and "data outage rate".

We solve the markov chains to determine the expected time to data loss (MTTDL). Then, we eliminate the death state (absorbing state) from the chain and solve it again to determine the steady-state probabilities of being in each non-death state. In conclusion, we characterize each logical unit by an annualized data loss rate and by the probability of being in each degraded mode and the duration of the degraded mode.

1. An annualized data loss rate:

   The annualized data loss rate (ADR) can be computed from the mean time to data loss (MTTDL) using the equation

$$ADR = 100 \times \frac{(HoursInOneYear)}{MTTDL} = 100 \times \frac{8760}{MTTDL}$$

2. A list of states or modes of operation. Each state is described by the number of failed and operational components. For each state, the probability of being in that state and the duration of that state (repair time) are specified as shown for an example state *s* below.

   **state s = (0,1,0)**

   **prob $p_s$ = 1.234E-4**

   **repair time $rt_s$ = 600 sec**

   The tuple (0,1,0) implies that there is one failure in the second component type (controller). The probability of being in this state is 1.234E-4 and the repair time is 10 minutes.

## 5.4 Assignment engine

In the assignment step, workload units are assigned to logical units with the objective of minimizing total cost.

The problem of assigning workload units to a set of devices (logical units) has been formulated as an instance of the discrete multiple-knapsack multiple-constraint problem. In principle, any approximate algorithm developed to solve this type of multiple-knapsack problem can be used to solve the assignment problem. [Shriver96] presents a formalization of the assignment problem. Qos goals are represented as constraint equations that have to hold for an assignment of a workload unit to a device to be acceptable. The assignment algorithm (solver) attempts to minimize some objective functions. The basic objective function that we are interested in here is the total cost of the system.

We first overview the constraints that have to be verified by the assignment solver. Next, we describe the "total cost" objective function and show how availability goals are incorporated in the function.

### *5.4.1 Assignment constraints*

An assignment *A* of workload units to logical units can be regarded as a mapping from the set of objects *W* to the set of logical units *L*.

1. **Reliability constraint**:

   Workload unit *W* can be assigned to logical unit *L* (*A(w)=l)* only if the data loss rate of the logical unit is less than or equal to the maximum data loss rate specified by the workload unit.

   $$DataLossRate(l) \leq MaxDataLossRate(w)$$

2. **Capacity constraint**:

   The size of the objects assigned to a logical unit is less than the capacity of the logical unit.

   $$\sum_{w,\, A(w) = l} Size(w) \leq Capacity(l)$$

3. **Performance constraints**:

   The performance constraint we deal with is simple. For each state (fault-free or degraded mode) of a logical unit, the sum of the throughputs (short-term transfer rate) of all the workload units assigned to a logical unit should be less than the throughput of the logical unit in that state. We use throughput to mean the number of requests per second that the logical unit can satisfy assuming a request stream with a certain average request size and random locality.

4. **Availability constraints**:

   The data outage duration and data outage rate of the device are less than the object's goals.

   $$DataOutageRate(l) \leq MaxDataOutageRate(w)$$
   $$DataOutageDuration(l) \leq MaxDataOutageDuration(w)$$

### *5.4.2 Objective function: minimizing total effective cost*

We introduce the notion of the "effective cost" of a storage system, which is a more appropriate measure of cost to use when configuring storage systems to meet availability requirements at minimal cost [Amiri96], [Golding94]. The effective cost of storage system is the "purchase cost" plus the cost resulting from the risk of data outages during the lifetime of the system.

For example, consider a storage system, which a cost of purchase $Cost_{purchase}$. Suppose that the system experiences outages of length $L_{outage}$ = 20 minutes, with an annualized data outage rate $R_{outage}$ =3%. The system is used to store the a data object which has a linear cost of downtime of $10,000 per minute, i.e. *CDO(20min)=$10,000*20=$200,000*. Then, the effective cost of the system over *n* years of usage is:

$$Cost_{eff} = Cost_{purchase} + \sum_{i=1}^{n} \frac{CDO(L_{outage}) \times R_{outage}}{(1+D)^{i}}$$

or equivalently

$$Cost_{eff} = Cost_{purchase} + \sum_{i=1}^{n} \frac{200,000 \times 0.03}{(1+0.1)^{i}}$$

Each term of the summation represents the cost of downtime during one year (the $i^{th}$ year). This cost model incorporates the discount rate *D* (which expresses the change in the value of money in terms of time). With a discount rate of 10%, a sum of *x* dollars today is worth $(1+0.1)^2 x$ dollars in two years.

We note that workload units are assigned to logical units with the assumption that admission policies (what workload units are serviced during degraded modes) are enforceable by the device or by the device manager. The motivation behind admission policies should be clear. When a device is in a degraded mode, it is desirable that it admits only a certain set of workloads for which it can meet the baseline requirements, rather than spread itself over the fault-free workload and end up not meeting any guarantees.

In the following, we describe the objective function of total effective cost. But first of all, we introduce some necessary notational definitions:

- We have already defined the assignment to be a mapping from the set of workload units *W* to the set of logical units *L*.

  *A(w) = l* means workload unit w has been assigned to logical unit *l*.

- We define the admission policy, denoted by *Adm*, as the mapping from the cross product set of logical units, workload units and states *LxWxS* to the set *{0,1}* where

$$Adm(l, w, s) = 1 \Leftrightarrow \begin{cases} A(w) = l, \text{ and} \\ w \text{ is admitted in state } s \text{ of unit } l \end{cases}$$

  In all other cases *Adm(l,w,s) = 0*. If workload unit *w* is not admitted in a certain state, then we refer to that state as a data outage state for workload unit *w*.

- *l* denotes a logical unit, we denote the number of modes (fault-free +degraded modes) with *modes(l)*. For each mode or state *s, s =1... modes(l),* we denote the mean duration of the mode (repair time) by $rt_s$ and the probability of occurrence of the mode by $p_s$.

- *w* denotes a workload unit, $CDO_w$ denotes the cost of downtime for workload unit *w,* which is a function of the downtime duration.

The total effective cost of the system is defined as the sum of the effective costs of the logical units used up by the assignment. The effective cost of a logical unit is the sum of the purchase cost of the logical unit and the sum of the cost of downtime for all the workload units asigned to the logical unit. The cost of downtime for a workload unit *w* assigned to a unit *l* is the sum of the products of the annualized data outage rate of each type of outage and the cost of the outage. A workload unit experiences a data outage in the states *s*, during which it is not admitted at unit *l*,

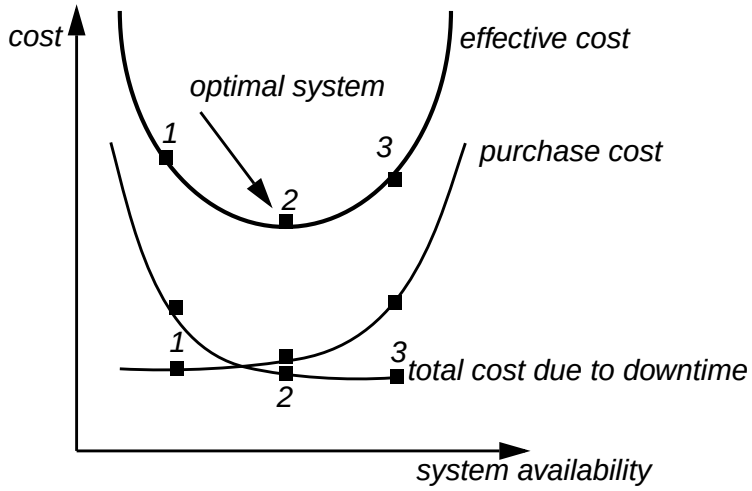


**Figure 8**. Cost of system versus availability. While purchase cost increases with availability, cost due to downtime obviously decreases for more available systems. Effective cost is the sum of the purchase cost and the cost due to downtime. Clearly, there is a point towards the middle where the total effective cost is minimized.

i.e. the states for which *Adm(l,s,w)=0*. Let *RO(s)* denote the annualized rate of occurrence of state *s* (The value of RO(s) is derived in Appendix A). Thus, for a workload unit w, the cost of downtime (data outages) over a period of *n* years is given by

$$Cost_{downtime}(w) = \sum_{s,\ Adm(l,s,w)=0} \left( \sum_{i=1}^{n} \frac{RO(s) \times CDO_w(rt_s)}{(1+D)^i} \right)$$

where *D* is the discount rate.

Now, we can express the total effective cost of the storage system as the cost of purchase of the devices and the cost of downtime of the workload units

$$Cost_{total} = \sum_{l,\ used(l)} Cost_{purchase}(l) + \sum_{l,\ used(l)} \left[ \sum_{w,\ A(w)=l} Cost_{downtime}(w) \right]$$

### *5.4.3 Assignment algorithm*

The assignment problem is hard because the discrete multiple-knapsack multiple constraint problem is NP-hard. The efficient algorithms proposed to solve this problem are approximate algorithms that use some heuristic to ensure that a solution that is *close enough* to optimal is reached within a reasonable time.

For this particular instance of the knapsack problem, we know that higher availability can be achieved via increased hardware redundancy and through the use of more reliable components. This comes at increased purchase cost. However, higher availability implies less data outages and therefore decreased losses due to downtime. Figure 8 graphs the effective cost of a logical unit (storage system) as a function of system “availability”. Our goal is to minimize the objective function of total effective cost. The heuristics employed by the algorithm are based on the following observations:

- Each workload unit taken independently, we can match it with the optimal logical unit. Namely, the logical unit which minimizes the total effective cost. For an arbitrary workload unit *w*, we denote the effective cost of the optimal logical unit (assuming the workload unit

is independently assigned) by *min_eff_cost(w)*. This cost denotes the sum of the purchase cost and the downtime cost for the logical unit that minimizes the effective cost for this particular workload unit, while meeting the reliability, capacity and availability constraints of the workload unit.

- Some workload units 'demand' high availability and will have to be matched with highly available 'expensive logical units'. Others are satisfied with cheaper logical units. We would like to allow for workload units to be assigned to better devices than they require if the units have to be purchased anyway (to satisfy a demanding workload unit). However, we would like to avoid scenarios where low demand workload units end up being assigned to expensive devices instead of demanding workload units.
- Often, some workload units will end up being assigned to logical units that have higher availability (and higher purchase cost) because the logical unit was required by some *demanding* workload unit, but that workload unit did not use up all the capacity. We define a measure of how much 'overmatched' a workload unit was, that we call workload unit *entropy*. The entropy of a workload unit *w* (that has been assigned to logical unit *l*) is defined as the ratio of the actual effective cost of the logical unit *l* (assuming it has been assigned *w* only) to the minimal effective cost of workload unit *w*. In other words,

$$entropy(w) = \frac{act\_eff\_cost(w)}{min\_eff\_cost(w)} = \frac{purchase\_cost(l)+cost\_of\_downtime(l,w)}{min\_eff\_cost(w)}$$

- One important observation is that given an initial assignment of workload units to logical units, we can identify the well-matched workload units by looking at their entropies. An entropy of one implies a well-matched workload unit (The workload unit was assigned to the logical unit to which it would have been assigned if it were the only workload unit). However, workload units with high entropies have been mismatched (notice the effective cost curve of Figure 8 is shaped like a valley. A workload unit has high entropy if it has been assigned to a logical unit corresponding to a point on either sides of the valley curve). Note that an assignment may have many workload units with high entropies while still being optimal (minimizing the total effective cost).
- Assuming we reached an initial assignment. We can identify the high entropy workload units and swap them with workload units that are better matches to that space.

A good heuristic is then to reach a decent intial assignment refine the assignment using information on how well things were matched and packed into the logical units. The algorithm is not fancy at this point as it has not been the focus of this work.

The algorithm goes as follows:

1. Greedy first-pass

   Iterating over the workload units in input order, assign a workload unit to the logical unit that will minimize total effective cost (purchase cost of all logical units + total cost of downtime for all workload units already assigned). The availability, reliability and performance constraints should hold of course.

2. Refinement passes: multiple passes to refine assignment

   Characterize the workload units according to the entropy measure.

   Remove (unassign) objects with high entropies from the logical units they have been assigned to and assign in their place objects that are better matched with that logical unit.

### 5.5 Future work

The approach we developed can be immediately extended to model logical units more accurately. Caches and Non-volatile RAM (NVRAM) can be included as an other component of a logical unit. Dependent failure rates of components (such as volatile caches failing in case of power failure, NVRAM failing in case of controller failure) can be handled within the developed framework. A lot of work remains to be done in several areas, including

1. Evaluating the approach and validating the tool: how optimal is the configuration? are the availability requirements met in practice?
2. Extending the concept of logical units to include weakly-consistent (and other dynamic) replication schemes. In principle, the degree of consistency of the replicas can be determined from the MWL availability requirement. Weakly-consistent replicas can be a much more cost-effective high-availability solution for certain applications than expensive fully redundant disk arrays.
3. Incorporating the impact of the s*torage management system itself* on the user-perceived object availability. Suppose there is a single storage manager (which performs functions such as object name resolution and mapping, device management, etc,...) and that manager is frequently unavailable. Then, the user-perceived availability of a certain object may be much worse than the availability of the logical unit on which the object is stored. Assuming that storage managers can be replicated, it is useful to determine what amount of replication necessary to meet the objects' availability requirements.

# 6 Conclusions

As storage system continue to grow, their configuration and administration will continue to be costly and poorly done. This is because the configuration space is large and involves complex trade-offs in cost, performance and availability. The attribute-managed storage project [Golding94] proposes to reduce the difficulty of this problem by having end-users specify the availability requirements of workload units to the storage system and let the system configure itself to meet these requirements. We demonstrate the feasibility of this proposition by describing an approach and developing a working tool to automatically design storage systems to meet the availability requirements of a large set of workload units. The tool automatically synthesizes all candidate storage logical units that match the input workload units and assesses their reliability, availability and performance via automatically generated Markov chains. An Assignment engine selects the appropriate storage logical units and determines the assignment of workload units to storage logical units so as to minimize total storage system cost.

We introduced a more appropriate cost measure that we call the effective cost of a storage system which incorporates the purchase cost of the system together with the cost (losses) incurred as a result of data unavailability due to system downtime.

# 7 References

[Allen96] Mark Allen. Disc drive reliability revealed. Hewlett-Packard internal report.

[Amiri96] Khalil Amiri and John Wilkes. Specifying data availability requirements. Hewlett-Packard Laboratories Technical Report HPL--SSP--96--16, August 1996.

[Bavuso87] Salvatore J. Bavuso, Joanne B. Dugan, Kishor S. Trivedi, Elizabeth M. Rothmann and W. Earl Smith. Analysis of fault-tolerant architectures using HARP. *IEEE Transactions on Reliability*, pages 176-185, Vol. R-36, No. 2, June 1987.

[Beaudry78] M. Danielle Beaudry. Performance-related reliability measures for computing systems. *IEEE Transactions on Computers*, pages 540-547, Vol. C-27, No. 6, June 1978.

[Fleischmann96] Marc Fleishmann. On high availability. Hewlett-Packard laboratories technical report HPL-CSL-96-x.

[Gibson93] Garth A. Gibson and David A. Patterson. Designing disk arrays for high data reliability. *Journal of Parallel and Distributed Computing*, **17**:4-27, 1993.

[Golding94] Richard Golding, Elizabeth Shriver, Tim Sullivan and John Wilkes. Attribute-managed storage. *Proceedings of the Workshop on Modelling and Specification of I/O*, San Antonio, Texas, October 1995.

[Hariri95] Salim Hariri and Hasan Mutlu. Hierarchical modelling of availability in distributed systems. *IEEE Transactions on Software Engineering*, Vol. 21, No. 1, January 1995.

[Holland92] Mark Holland Garth A. Gibson. Parity declustering for continuous operation in redundant disk arrays. *Proceedings of the 5th International Conference on Architectural Support for Programming Languages and Operating Systems* (Boston, MA). Published as *Computer Architecture News*, **20**(special issue):23-35, 12-15 October 1992.

[Malhotra93] Manish Malhotra and Kishor S. Trivedi. Reliability analysis of redundant arrays of independent disks. *Journal of Parallel and Distributed Computing*, **17**:146-151, 1993.

[Meyer82] John F. Meyer. On evaluating the performability of degradable computing systems. *IEEE Transactions on Computers*, pages 720-731, Vol. C-29, No. 8, Ausgust 1980.

[Ng90] Spencer W. Ng. Sparing for a redundant disk array. IBM research report RJ 7621, August 1990.

[Patterson88] David A. Patterson, Garth A. Gibson and Randy H. Katz. A case for redundant arrays of independent disks (RAID). *Proceedings of SIGMOD*. (Chicago, Illinois), 1-3 June 1988.

[Pattipati93] Krishna R. Pattipati, Yong Li and Henk A. P. Blom. A unified framework for the performability evaluation of fault-tolerant computer systems. *IEEE Transactions on Computers*, Vol. 42, No. 3, March 1993.

[Savage95] Stefan Savage and John Wilkes. AFRAID---A Frequently Redundant Array of Independent Disks. *Proceedings of Winter 96 USENIX Conference* (San Diego, CA), January 1996.

[Shculze89] Martin Schulze, Garth A. Gibson, Randy Katz and David Patterson. How reliable is a RAID? *Spring COMPCON'89* (San Fancisco), pages 118-23. IEEE, March 1989.

[Siewiorek92] Daniel P. Siewiorek and Robert S. Swarz. *Reliable computer systems: design and evaluation*. Digital Press, Second edition, 1992.

[Wilkes90] John Wilkes and Raymie Stata. Specifying data availability for muti-device file systems. *Proceedings of the 4th ACM SIGOPS European Workshop*, Bologna (3-5 September 1990).

# 8 Appendix A: availability computations

We model the availability of logical units via a Markov chain which we solve to get the probability of the logical unit being in each fault-free or degraded mode state. When assigning a workload unit to a logical unit, we need to evaluate the availability constraints to make sure they hold. Also,

we need to determine the cost of downtime for the workload unit if it were assigned to the logical unit.

This section shows how to derive the maximum outage duration, the maximum outage rate and the cost of downtime for a workload unit assigned to a logical unit from the state probabilities and durations of the unit.

Suppose *w* is a workload unit and *l* is a logical unit with states *s=1...S*. Without loss of generality, we assume is admitted at the logical unit *l* in the first *M* states *s=1...M*, so that states *M+1,...,S* are considered outages states for workload unit *w*.

Furthermore, let's denote the steady-state probability of logical unit *l* being in state *s* by $p_s$, and let's denote the duration of state *s* by $rt_s$.

All the values we use in this analysis are mean values (state duration).

1. Deriving the longest and mean outage durations if *w* were assigned to *l*

   The data outage states for workload unit w are states *M+1,...,S* during which the workload unit is not admitted at logical unit *l*. The maximum outage duration is therefore the duration of the longest state.

   Let the longest outage state be state *m*, then the mean duration of state *m* is given by $rt_m$. So formally, the longest mean outage duration is given by

$$LOD = MAX(rt_s), \text{ for all outage states } s = M+1,...,S$$

   Alternatively, we can use the mean outage duration as the availability requirement. The mean outage duration for workload unit *w* when assigned to logical unit *l* can be computed by averaging the duration of all outage states. The mean outage duration (MOD) is given by

$$MOD(w) = \frac{\sum_{i=M+1}^{S} p_i}{\sum_{i=M+1}^{s} \frac{p_i}{rt_i}}$$

2. Deriving the mean outage rate for *w*

   The rate of outages as seen by a workload unit *w* is the sum of the rates of all outage states for *w*. Suppose *m* is an outage state, the probability of being in state *m* is $p_m$ and the mean duration of state *m is* $rt_m$, then the annualized rate of occurrence *RO(m)* of state *m* related to the mean time between outages as discussed in section 2.4. The mean time between outages (occurrences of state *m*) is given by the ratio of the duration of the state to the probability of its occurrence, thus

$$RO(m) = 100 \times \frac{HoursInOneYear}{\frac{rt_m}{p_m}}$$

Therefore, the total outage rate for workload unit *w* is given by the sum of the rates of all the outage states (the outage states are states *M+1,...,S)*

$$RO(w) = \sum_{i=M+1}^{S} RO(m) = \sum_{i=M+1}^{S} 100 \times \frac{HoursInOneYear}{\frac{rt_m}{p_m}}$$

3. Deriving the mean cost of downtime for *w*

The cost of downtime for workload unit *w* over *n* years is given by the sum of the costs of all the outages experienced by the workload unit over the *n* years. The cost of date outages over the course of one year is equal to the product of the rate of outages by the cost of each outage. Formally, if D is the annual discount rate then the cost of downtime for workload unit *w* over the course of *n* years is

$$Cost_{downtime}(w) = \sum_{s=M+1}^{S} \left( \sum_{i=1}^{n} \frac{RO(s) \times CDO_w(rt_s)}{(1+D)^i} \right)$$